

Model-Based Specification

Learning Objective

... Formal specification of software by
developing a *mathematical model of the system*.

Frederick T Sheldon

Assistant Professor of Computer Science

Washington State University

Objectives



To introduce an approach to formal specification based on mathematical system models

To present some features of the Z specification language

To illustrate the usefulness of Z by describing small examples

To show how *Z schemas may be used to develop incremental specifications*

Topics covered

Z schemas

The Z specification process

Specifying ordered collections

Model-based specification

Defines a model of a system using well-understood mathematical entities such as sets and functions.

The state of the system is not hidden (unlike algebraic specification).

State changes are straightforward to define.

VDM and Z are the most widely used model-based specification languages.

Z as a specification language

Based on *typed set theory*

Probably now the most widely-used
specification language

Includes **schemas**, an *effective low-level*
structuring facility

Schemas are specification building blocks

Graphical presentation of schemas make Z
specifications easier to understand

Z schemas



Introduce specification entities and defines
invariant predicates over these entities

A schema includes

- A name identifying the schema

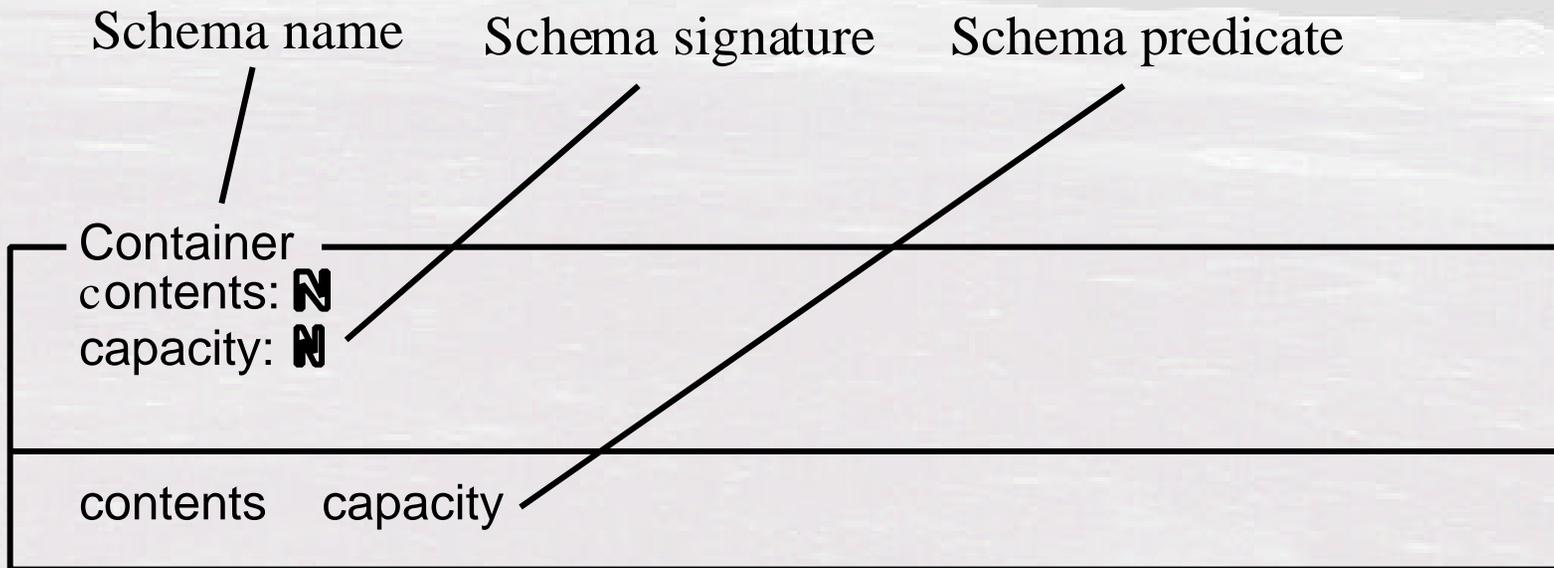
- A signature introducing entities and their types

- A predicate part defining invariants over these entities

Schemas can be included in other schemas **and**
may act as type definitions

Names are local to schemas

Z schema highlighting



An indicator specification

Indicator

light: {off, on}

reading: **N**

danger_level: **N**

light = on

reading

danger_level

Storage tank specification

Storage_tank

Container
Indicator

reading = contents
capacity = 5000
danger_level = 50

Full specification of a storage tank

Storage_tank

contents: **N**

capacity: **N**

reading: **N**

danger_level: **N**

light: {off, on}

contents capacity

light = on reading danger_level

reading = contents

capacity = 5000

danger_level = 50

Z conventions

A variable name decorated with a quote mark (N') represents the value of the state variable N after an operation

A schema name decorated with a quote mark introduces the dashed values of all names defined in the schema

A variable name decorated with a ! represents an output

Z conventions

A variable name decorated with a ? represents an input

A schema name prefixed by the Greek letter Xi (Ξ) means that the defined operation does not change the values of state variables

A schema name prefixed by the Greek letter Delta (Δ) means that the operation changes some or all of the state variables introduced in that schema

Operation specification

Operations may be specified incrementally as separate schema then the schema combined to produce the complete specification

Define the ‘normal’ operation as a schema

Define schemas for exceptional situations

Combine all schemas using the disjunction (or) operator

A partial specification of a fill operation

Fill-OK

Storage_tank
amount?: \mathbb{N}

contents + amount? capacity
contents' = contents + amount?

Storage tank fill operation

OverFill

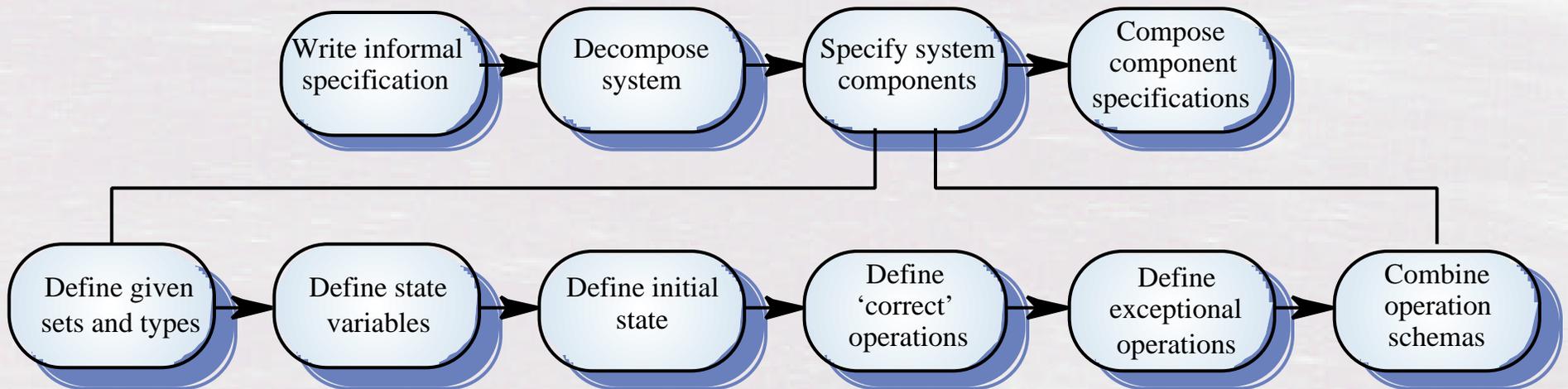
Storage-tank
amount?: **N**
r!: seq CHAR

capacity < contents + amount?
r! = "Insufficient tank capacity – Fill cancelled"

Fill

Fill-OK \vee OverFill

The Z specification process



Data dictionary specification

Data dictionary, introduced in Chapter 6, will be used as an example. This is part of a CASE system and is used to keep track of system names

Data dictionary structure

Item name

Description

Type. Assume in these examples that the allowed types are those used in semantic data models

Creation date

Given sets



Z does not require everything to be defined at specification time

Some entities may be ‘given’ and defined later

The first stage in the specification process is to introduce these given sets

[NAME, DATE]

We don’t care about these representations at this stage

Type definitions

**There are a number of built-in types
(such as INTEGER) in Z**

Other types may be defined by enumeration

Sem_model_types = { relation, entity, attribute }

»

Schemas may also be used for type definition

The predicates serve as constraints on the type

Specification using functions

A function is a *mapping* from an input value to an output value

$$\text{SmallSquare} = \{1, 2, 3, 4, 5, 6, 7\} \rightarrow \{1, 4, 9, 16, 25, 36, 49\}$$

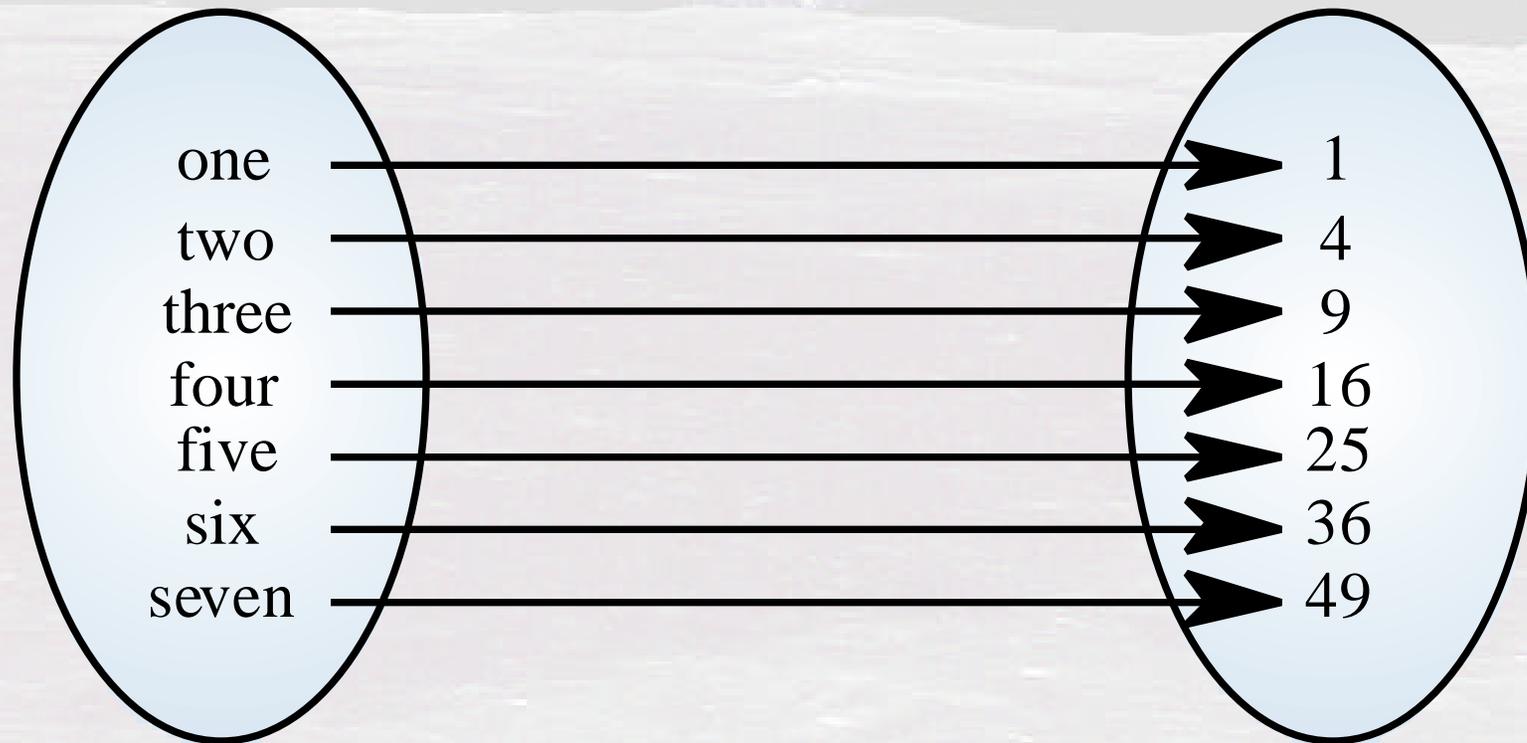
The domain of a function is the *set of inputs* over which the function has a defined result

$$\text{dom SmallSquare} = \{1, 2, 3, 4, 5, 6, 7\}$$

The range of a function is the *set of results* which the function can produce

$$\text{rng SmallSquare} = \{1, 4, 9, 16, 25, 36, 49\}$$

The function SmallSquare



Domain (SmallSquare)

Range (SmallSquare)

Data dictionary modeling

A data dictionary may be thought of as a mapping from a name (the key) to a value (the description in the dictionary)

Operations are

Add. Makes a new entry in the dictionary or replaces an existing entry

Lookup. Given a name, returns the description.

Delete. Deletes an entry from the dictionary

Replace. Replaces the information associated with an entry

Data dictionary entry

DataDictionaryEntry

entry: NAME

desc: seq char

type: Sem_model_types

creation_date: DATE

#description 2000

Data dictionary as a function

```
DataDictionary  
DataDictionaryEntry  
ddict: NAME ▸ {DataDictionaryEntry}
```

Data dictionary - initial state

Init-DataDictionary

DataDictionary'

ddict' = \emptyset

Add and lookup operations

Add_OK

DataDictionary
name?: NAME
entry?: DataDictionaryEntry

name? dom ddict
ddict' = ddict {name?, entry?}

Lookup_OK

DataDictionary
name?: NAME
entry!: DataDictionaryEntry

name? dom ddict
entry! = ddict (name?)

Add and lookup operations

Add_Error

DataDictionary
name?: NAME
error!: seq char

name? dom ddict
error! = "Name already in dictionary"

Lookup_Error

DataDictionary
name?: NAME
error!: seq char

name? dom ddict
error! = "Name not in dictionary"

Function over-riding operator

ReplaceEntry uses the function overriding operator (written \oplus). This adds a new entry or replaces an existing entry.

phone = { Ian 3390, Ray 3392, Steve 3427 }

The domain of phone is { Ian, Ray, Steve } and the range is { 3390, 3392, 3427 }.

newphone = { Steve 3386, Ron 3427 }

phone \oplus newphone = { Ian 3390, Ray 3392, Steve 3386, Ron 3427 }

Replace operation

Replace_OK

DataDictionary
name?: NAME
entry?: DataDictionaryEntry

name? dom ddict
ddict' {name? | entry?}

Deleting an entry

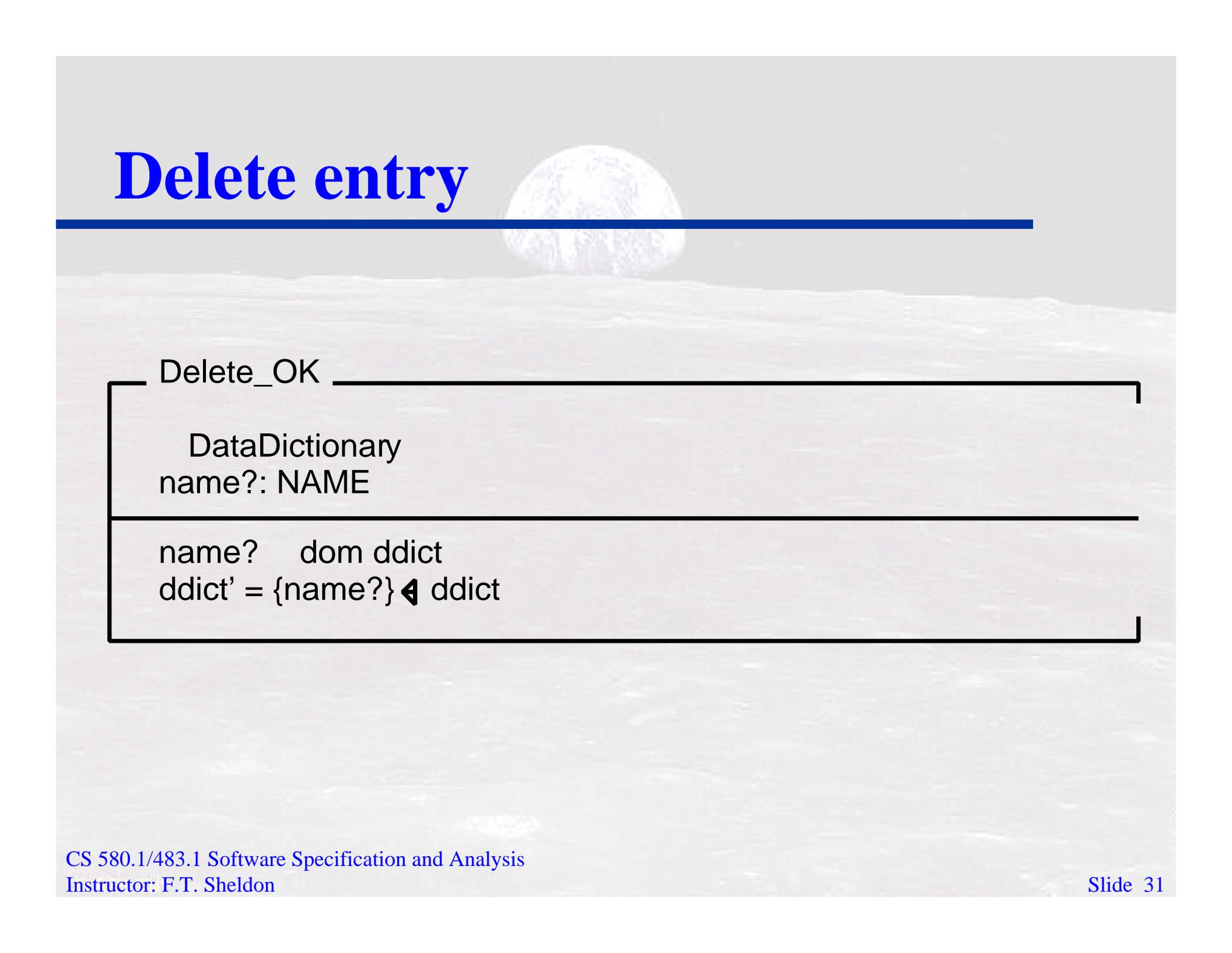
Uses the domain subtraction operator (written \setminus) which, given a name, removes that name from the domain of the function

phone = { Ian \rightarrow 3390, Ray \rightarrow 3392, Steve \rightarrow 3427 }

{Ian} \setminus phone

{Ray \rightarrow 3392, Steve \rightarrow 3427 }

Delete entry



Delete_OK

DataDictionary
name?: NAME

name? dom ddict
ddict' = {name?} ◀ ddict

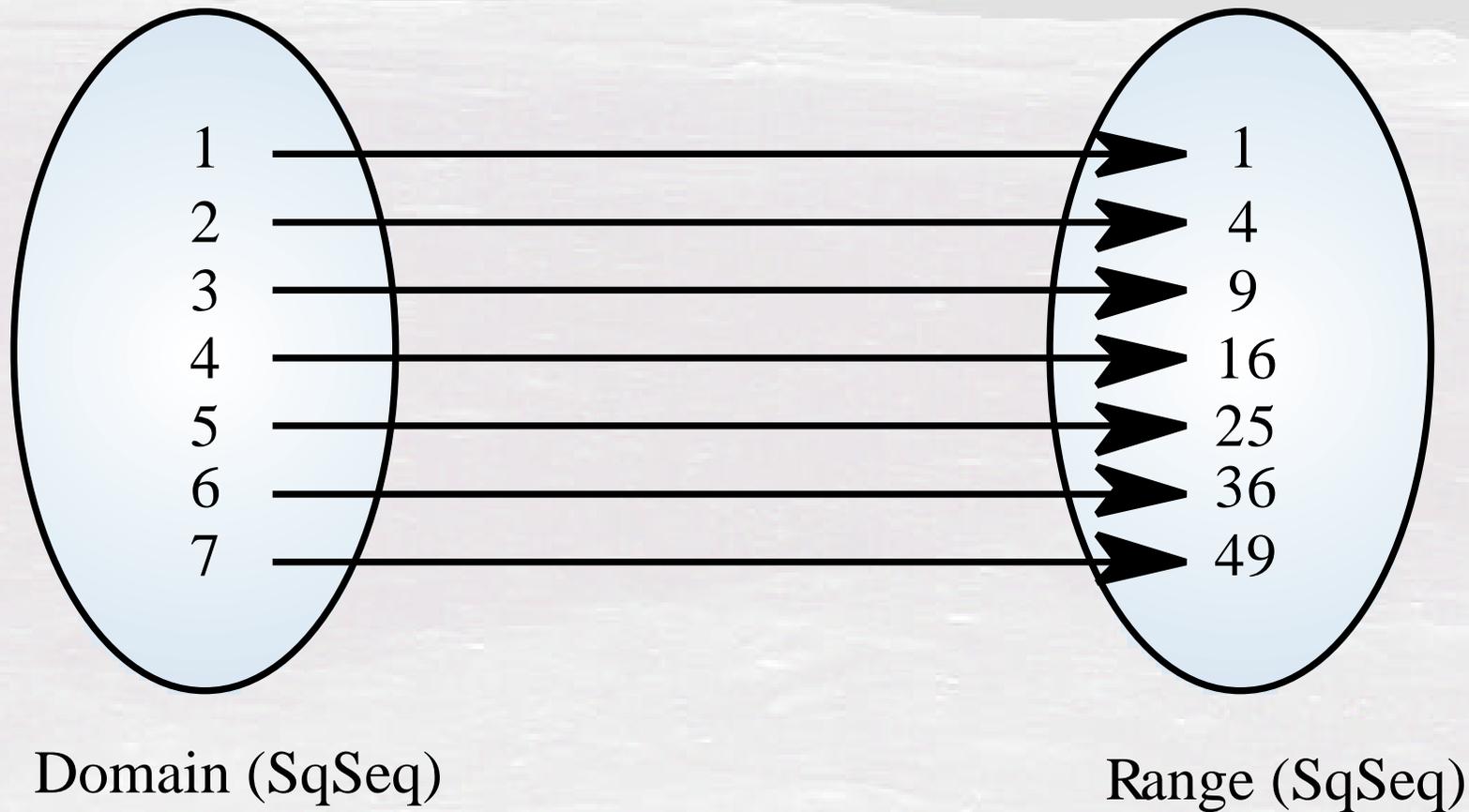
Specifying ordered collections

Specification using functions does not allow ordering to be specified

Sequences are used for specifying ordered collections

A sequence is a mapping from consecutive integers to associated values

A Z sequence



Data dictionary extract operation



The **Extract** operation extracts from the data dictionary all those entries whose type is the same as the type input to the operation

The *extracted list* is presented in alphabetical order

A **sequence** is used to specify the **ordered** output of **Extract**

The Extract operation

Extract

DataDictionary

rep!: seq {DataDictionaryEntry}

in_type?: Sem_model_types

$n : \text{dom ddict} \bullet \text{ddict}(n). \text{type} = \text{in_type?} \quad \text{ddict}(n) \quad \text{rng rep!}$

$i : 1 \quad i \quad \#\text{rep!} \bullet \text{rep!}(i). \text{type} = \text{in_type?}$

$i : 1 \quad i \quad \#\text{rep!} \bullet \text{rep!}(i) \quad \text{rng ddict}$

$i, j : \text{dom rep!} \bullet (i < j) \quad \text{rep.name}(i) <_{\text{NAME}} \text{rep.name}(j)$

Extract predicate

For all entries in the data dictionary whose type is **in_type?**, there is an entry in the output sequence

The type of all members of the output sequence is **in_type?**

All members of the output sequence are members of the range of **ddict**

The output sequence is ordered

Data dictionary specification

The_Data_Dictionary

DataDictionary

Init-DataDictionary

Add

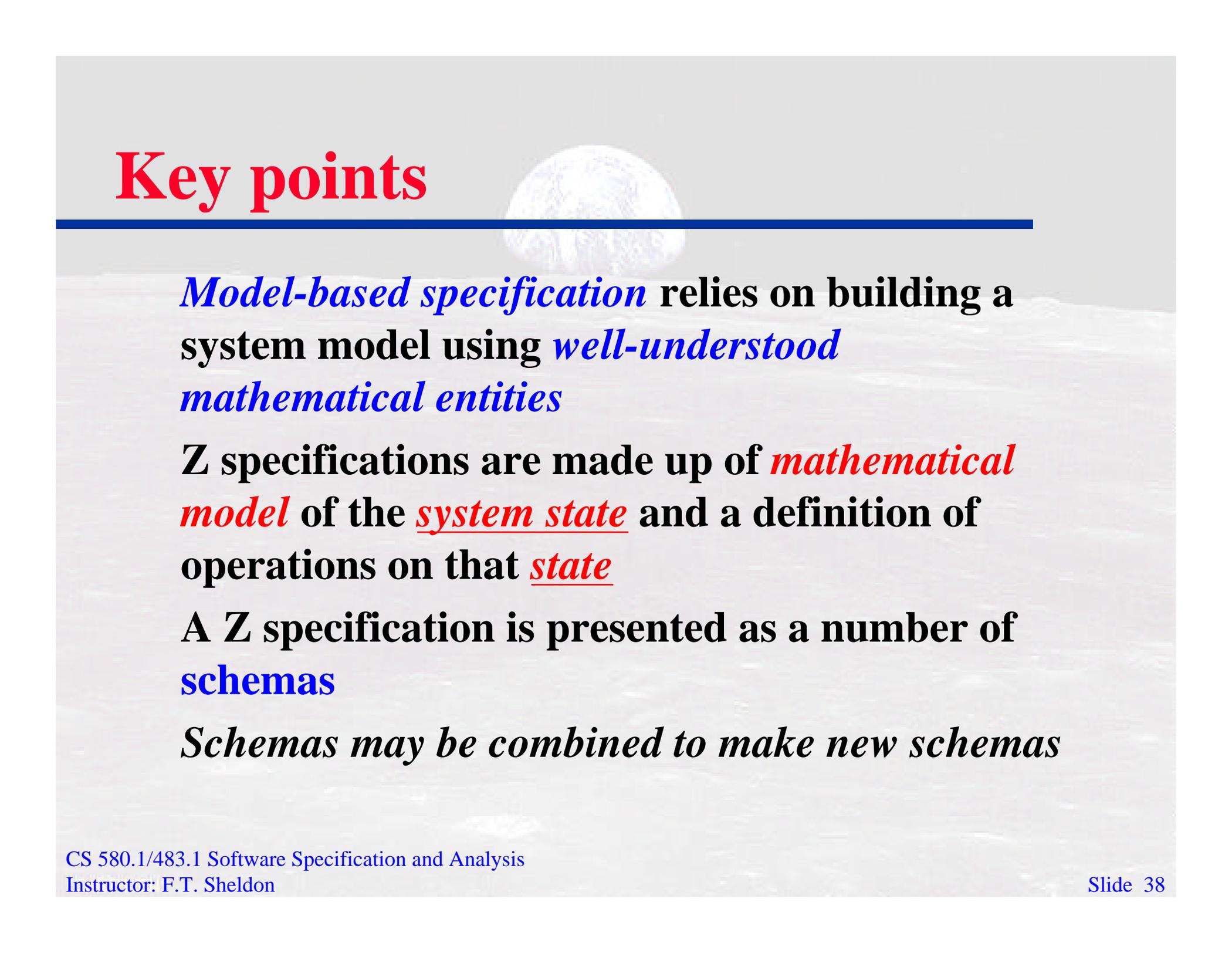
Lookup

Delete

Replace

Extract

Key points



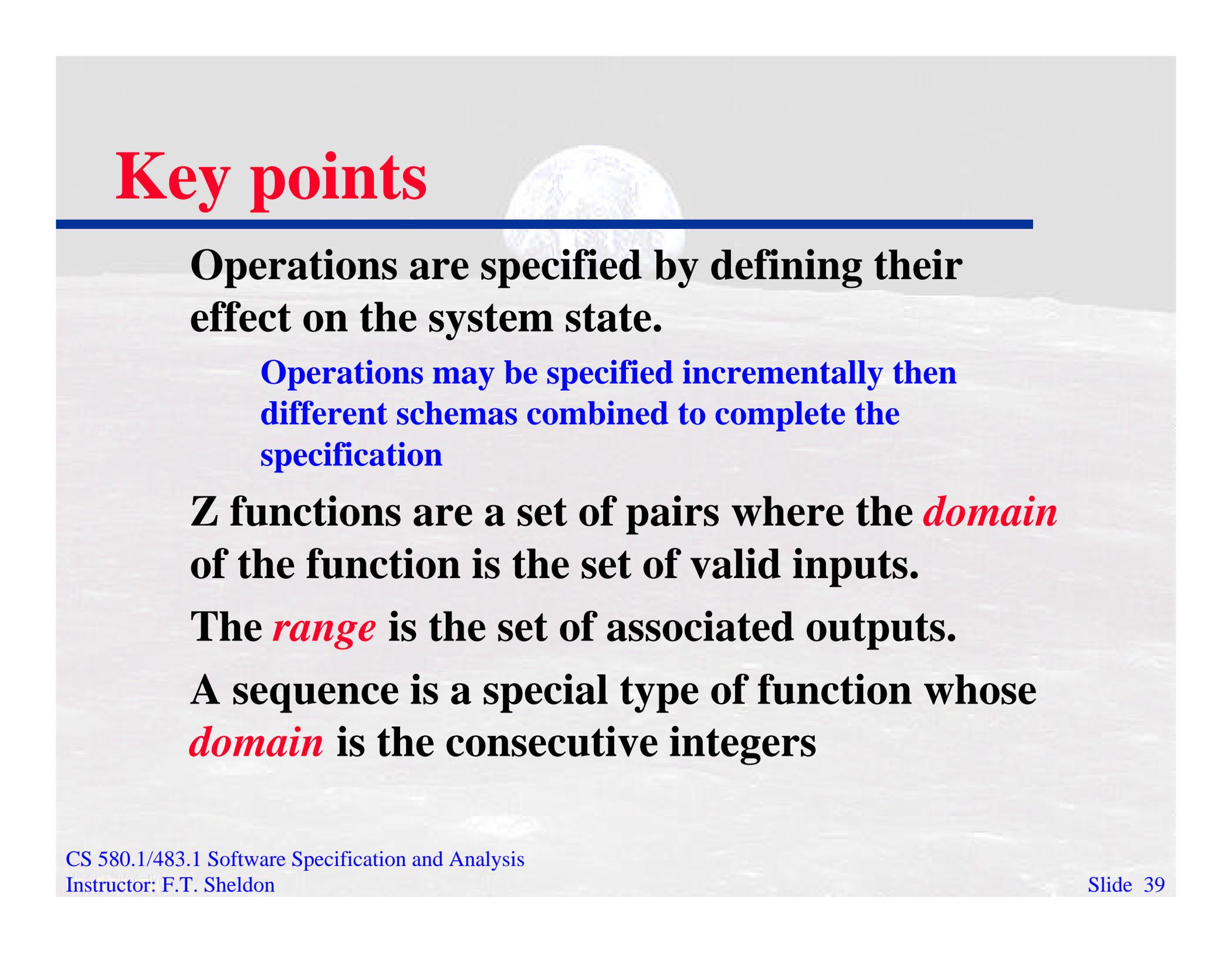
Model-based specification relies on building a system model using *well-understood mathematical entities*

Z specifications are made up of *mathematical model* of the *system state* and a definition of operations on that *state*

A Z specification is presented as a number of *schemas*

Schemas may be combined to make new schemas

Key points



Operations are specified by defining their effect on the system state.

Operations may be specified incrementally then different schemas combined to complete the specification

Z functions are a set of pairs where the *domain* of the function is the set of valid inputs.

The *range* is the set of associated outputs.

A sequence is a special type of function whose *domain* is the consecutive integers